背景

对软件开发者来说，参数校验让人深感头疼。在项目中，难免需要对参数进行一些参数正确性的校验，这些校验出现在业务代码中，让我们的业务代码显得臃肿，而且，频繁地编写这类参数校验代码很无聊。Hibernate Validator框架刚好解决了这些问题，可以很优雅的方式实现参数的校验，让业务代码和校验逻辑分开,不再编写重复的校验逻辑。

但Hibernate Validator仅仅解决的是静态参数的校验，如字符串的长度、数值的大小等，对于动态参数校验则无能为力了。

```
@NotBlank(message = "??id????")
private String accountId;

@NotBlank(message = "??????")
private String nickname;

@NotNull(message = "??????")
@Pattern(regexp = "^M$|^F", message = "?????")
private String sex;

@NotNull(message = "??????")
private Integer age;
```

动态参数校验是指在程序运行时，根据参数的实际值来决定程序是要抛出异常还是继续运行。

动态参数校验实战

Spring在提供一个强大的应用开发框架的同时也提供了很多优秀的开发工具类，合理的运用这些工具，将有助于提高开发效率、增强代码质量。Assert断言工具类，通常用于数据合法性检查，在JAVA编程中，通常会编写如下代码：

```
if (name == null || name.equls("")) {
 throw new IllegalArgumentException("参数错误!");
}
```

修改成Assert的方法：

```
String txt=null;

Assert.hasText(txt, "text must not be empty");
```

抛出异常：

```
Exception in thread "main" java.lang.IllegalArgumentException: text must not be empty
```

Spring 提供的 Assert 类拥有众多按规则对方法入参进行断言的方法，可以满足大部分方法入参检测的要求。其它用法还有类似：

```
public static void main(String[] args) {

int i=1;

Assert.isTrue(i > 0, "The value must be greater than zero");

String txt=null;

Assert.hasText(txt, "text must not be empty");


Boolean flag=true;

Assert.state(flag, "state must be true");


Integer clazz=null;

Assert.isNull(clazz, "The class must be null");

Assert.notNull(clazz, "The class must not be null");
```

```
    }
```

小结

可见使用 Spring 的 Assert
替代自编码实现的入参检测逻辑后，方法的简洁性得到了不少的提高。Assert 不依赖于 Spring 容器，您可以大胆地在自己的应用中使用这个工具类.